Abstract

MATLAB is an array language, initially popular for rapid prototyping, but is now being increasingly used to develop production code for numerical and scientific applications. Typical MATLAB programs have abundant data parallelism. These programs also have control flow dominated scalar regions that have an impact on the program's execution time. Today's computer systems have tremendous computing power in the form of traditional CPU cores and also throughput-oriented accelerators such as graphics processing units (GPUs). Thus, an approach that maps the control flow dominated regions of a MATLAB program to the CPU and the data parallel regions to the GPU can significantly improve program performance.

In this work, we present the design and implementation of MEGHA, a compiler that automatically compiles MATLAB programs to enable synergistic execution on heterogeneous processors. Our solution is fully automated and does not require programmer input for identifying data parallel regions. Our compiler identifies data parallel regions of the program and composes them into kernels. The kernel composition step eliminates a number of intermediate arrays which are otherwise required and also reduces the size of the scheduling and mapping problem the compiler needs to solve subsequently. The problem of combining statements into kernels is formulated as a constrained graph clustering problem. Heuristics are presented to map identified kernels to either the CPU or GPU so that kernel execution on the CPU and the GPU happens synergistically, and the amount of data transfer needed is minimized. A heuristic technique to ensure that memory accesses on the CPU exploit locality and those on the GPU are coalesced is also presented. In order to ensure that data transfers required for dependences across basic blocks are performed, we propose a data flow analysis step and an edge-splitting

strategy. Thus our compiler automatically handles kernel composition, mapping of kernels to CPU and GPU, scheduling and insertion of required data transfers.

Additionally, we address the problem of identifying what variables can coexist in GPU memory simultaneously under the GPU memory constraints. We formulate this problem as that of identifying maximal cliques in an interference graph. We approximate the interference graph using an interval graph and develop an efficient algorithm to solve the problem. Furthermore, we present two program transformations that optimize memory accesses on the GPU using the software managed scratchpad memory available in GPUs.

We have prototyped the proposed compiler using the Octave system. Our experiments using this implementation show a geometric mean speedup of 12X on the GeForce 8800 GTS and 29.2X on the Tesla S1070 over baseline MATLAB execution for data parallel benchmarks. Experiments also reveal that our method provides up to 10X speedup over hand written GPUmat versions of the benchmarks. Our method also provides a speedup of 5.3X on the GeForce 8800 GTS and 13.8X on the Tesla S1070 compared to compiled MATLAB code running on the CPU.